MINI PROJECT 5 – ALGEBRAIC DATA TYPES

BY: DAVI BORGES, NATHAN CADER, SIRISH GURUNG, EMILEE ZYGMUNT, DEVIN HART

1. PARSER: GRAMMAR AND PARSE STRATEGY

- GRAMMAR
  - {DATA <TYPENAME> {VARIANT1 ARG1 ARG2 ... } {VARIANT2} ... }
  - {MATCH {EXPRESSION} {CLAUSE1 ARG1 ARG2 => RESULT1} {CLAUSE2 => RESULT2} ... }
- STRATEGY
  - EXTRACT THE DATA TYPE'S NAME PLUS THE NAMES AND ARGUMENTS OF ALL VARIANTS FROM A DATA EXPRESSION
  - EXTRACT THE SCRUTINEE EXPRESSION ALONG WITH ALL CLAUSE NAMES AND ARGUMENTS FROM A MATCH EXPRESSION
  - PARSE VARIANTS AND CLAUSES TO VARIANTE AND CLAUSEE, WHICH CONTAIN THE EXTRACTED INFORMATION
  - PARSE MAIN FUNCTIONS TO DATAE AND MATCHE RESPECTIVELY

2. AST

- EXP (EXPRESSION) IS THE MAIN AST TYPE, WITH EACH LANGUAGE FEATURE REPRESENTED AS ITS OWN NODE SUCH AS NUME, VARE, LAME, APPE, AND IFE
- DATAE STORES THE DATA TYPE NAME AND A LIST OF VARIANTE, WHERE EACH VARIANT CONTAINS ITS CONSTRUCTOR NAME AND FIELD NAMES
- CONSTRUCTORE STORES THE CONSTRUCTOR NAME AND A LIST OF ARGUMENT EXPRESSIONS USED TO BUILD TAGGED VALUES
- MATCHE STORES THE EXPRESSION BEING MATCHED AND A LIST OF CLAUSEE, WHERE EACH CLAUSE CONTAINS THE CONSTRUCTOR NAME, BOUND VARIABLES, AND BODY EXPRESSION

3. VALUES: REPRESENTATION OF DATA

- TAGGED VALUES LET THE LANGUAGE STORE USER-DEFINED DATA WITH MEANING, NOT JUST RAW NUMBERS OR SIMPLE TEXT.
- WHY YOU SHOULD USE IT?
  - A RAW VALUE DOES NOT SAY WHAT KIND OF VALUE IT IS.
  - MAKES POSSIBLE PATTERN MATCHING POSSIBLES. MAKING THE TAG LABEL THE DATA, AND THE FIELDS STORING THE CONTENTS.
- QUICK EXAMPLES:
  - {TAG FIELD}
  - {CARPLATE ABC123} -> (TAGV 'CARPLATE (LIST ABC123))
  - {ZIPCODE 33884} -> (TAGV 'ZIPCODE (LIST 33884))
  - {NOTHING} -> (TAGV 'NOTHING EMPTY)

4. CONSTRUCTORS IN THE ENVIRONMENT

- **ENVIRONMENT PRE-LOADING:** BEFORE EXECUTION, DATAE DEFINITIONS ARE PARSED, AND VARIANTS ARE INJECTED DIRECTLY INTO THE GLOBAL ENVIRONMENT AS BASE TAGV (TAGGED VALUES) WITH EMPTY FIELD LISTS.
- **UNIFIED APPLICATION:** CONSTRUCTORS ARE TREATED AS STANDARD FUNCTIONS; THE PARSER CONVERTS CONSTRUCTOR CALLS INTO STANDARD FUNCTION APPLICATIONS (APPE).
- **ARGUMENT ACCUMULATION:** WITHIN THE INTERPRETER'S APPE LOGIC, APPLYING AN ARGUMENT TO A TAGV AUTOMATICALLY APPENDS THAT ARGUMENT TO ITS FIELD LIST.
- **NATIVE CONSTRUCTION:** THIS APPROACH ALLOWS MULTI-FIELD DATA STRUCTURES (E.G., `{RECT 10 20}`) TO NATIVELY ACCUMULATE ARGUMENTS ONE BY ONE UNTIL THE FINAL TAGGED VALUE IS BUILT.

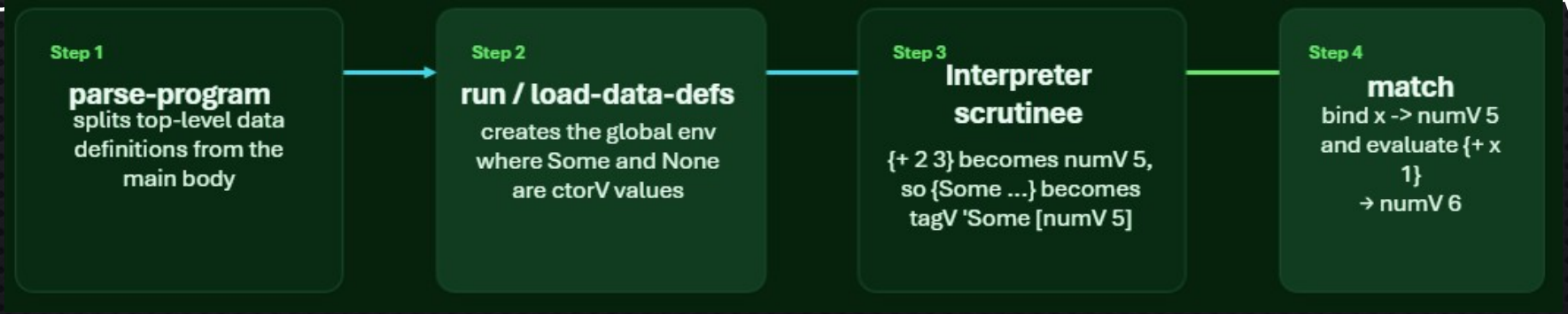
5. INTERPRETER / MATCH

- Interpreter is the main dispatcher for the whole language. Turns syntax into value objects. evaluation for numbers, variables, functions, constructors, conditionals
- The matcher helps match patterns to tagged data, then binds pattern variables and interprets the matching body to the environment. Only works after constructor (ctorV) application has produced a tagged value (tagV)

6. TEST AND REFLECTION

```
(run-surface
 (list
  `{data Maybe {None} {Some x}}
  `{match {Some {+ 2 3}}
      {Some x => {+ x 1}}
      {None => 0}}))
```

- ADDED TESTS FOR CONSTRUCTORS, MATCHING, AND ERROR CASES
- VERIFIED SUPPORT FOR ZERO-FIELD, ONE-FIELD, AND MULTI-FIELD VARIANTS
- EXTENDED THE LANGUAGE WITH DATA, CONSTRUCTORS, AND MATCH
- ADDED TAGGED RUNTIME VALUES WITH TAGV
- CONFIRMED THAT PATTERN MATCHING SAFELY BINDS FIELDS BY VARIANT


Step 1
parse-program
splits top-level data definitions from the main body
Step 2
run / load-data-defs
creates the global env where Some and None are ctorV values
Step 3
Interpreter scrutinee
{+ 2 3} becomes numV 5, so {Some ...} becomes tagV 'Some [numV 5]
Step 4
match
bind x -> numV 5 and evaluate {+ x 1}
→ numV 6